Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (currently amended): An apparatus for pseudo-random testing binary emulation, comprising:

a random code generator ~~that generates~~ capable of generating an initial machine state and a binary instruction sequence;

a native architecture execution engine ~~that executes~~ capable of executing the binary instruction sequence to produce a first final state ~~S1~~;

a target architecture execution engine ~~that executes~~ capable of executing the binary instruction sequence concurrently with the native architecture execution engine to produce a second final state ~~S2~~, the target architecture execution engine comprising a binary emulator ~~that emulates~~ capable of emulating the binary instruction sequence according to the target architecture; and

a verification engine ~~that compares~~ capable of comparing the first final state ~~S1~~ and the second final state ~~S2~~, wherein when the first final state ~~S1~~ and the second final state ~~S2~~ do not match, an emulation failure has occurred;

wherein an emulated binary instruction sequence that generates the emulation failure is a short sequence of binary instructions, enabling the emulation failure to be determined at a machine instruction level.

2. (original): The apparatus of claim 1, wherein the native and the target architecture execution engines communicate using machine-to-machine communications protocols.

3. (original): The apparatus of claim 2, wherein the communications protocol is a TCP/IP protocol.

4. (original): The apparatus of claim 1, wherein when the emulation failure occurs, information related to the emulation failure is written to a file.

5. (canceled).

6. (original): The apparatus of claim 1, wherein the target platform is a hardware embodiment.

7. (original): The apparatus of claim 1, wherein the random code generator comprises a probability file comprising probability values for each instruction in a native instructions set architecture (ISA).

8. (original): The apparatus of claim 7, wherein the probability values in the probability file are user-generated.

9. (currently amended): A method for pseudo-random testing of binary emulation, comprising:

generating a pseudo-random binary instruction sequence;

generating an initial machine state;

initializing a native machine according to the initial machine state;

executing the binary instruction sequence on the native machine to produce a first final state ~~S1~~;

initializing a target machine according to the initial machine state;

emulating the binary instruction sequence on the target machine;

executing the emulated binary instruction sequence to produce a second final state ~~S2~~ concurrently with the step of executing the binary instruction sequence on the native machine; ~~and~~

comparing the first final state ~~S1~~ and the second final state ~~S2~~ to determine an emulation error; and

enabling the emulation error to be determined at a machine instruction level.

10. (currently amended): The method of claim 9, further comprising:

communicating the initial machine state and the binary instruction sequence to the target machine using a machine-to-machine communication protocol; and

communicating the second final state ~~S2~~ to the native machine using the machine-to-machine communication protocol.

11. (original): The method of claim 10, wherein the machine-to-machine communication protocol is a TCP/IP protocol.

12. (currently amended): The method of claim 9, further comprising storing information related to the emulation failure, the information comprising the initial machine state, the binary instruction sequence and the first and second final states ~~S1 and S2~~.

13. (original): The method of claim 9, wherein the binary emulation is executed on a simulator.

14. (original): The method of claim 9, wherein the binary emulation is executed on a hardware device.

15. (original): The method of claim 9, further comprising:

assigning a probability value to each binary instruction in a native instruction set architecture (ISA);

pseudo-randomly generating a probability value; and

selecting the binary instruction sequence based on the probability value.

16. (currently amended): A method for verifying cross-architecture emulation, comprising:

randomly generating a native instruction sequence and an initial machine state;

providing the native instruction sequence and the initial machine state to a first platform having a first instruction set architecture (ISA);

providing the native instruction sequence to a second platform having a second ISA;

emulating the native instruction sequence according to the second ISA;

executing the native instruction sequence in the first platform, the execution providing a first final state ~~S1~~;

executing the emulated native instruction sequence on the second platform, the execution providing a second final state ~~S2~~ concurrently with the step of executing the native instruction sequence; ~~and~~

comparing the first and second final states ~~S1 and S2~~ to determine an emulation failure; and

enabling the emulation failure to be determined at a machine instruction level.

17. (original): The method of claim 16, wherein the native instruction sequence is randomly generated from a set of all instructions in the first ISA.

18. (original): The method of claim 16, wherein the random generation is based on a user-defined value assigned to each instruction in the first ISA.

19. (canceled).

20. (original): The method of claim 16, wherein the emulation is completed on a binary instruction emulator operating on the second platform.

21. (new): A computer readable medium providing instructions for pseudo-random testing binary emulation, the instructions comprising:

generating an initial machine state and a binary instruction sequence;

executing the binary instruction sequence to produce a first final state;

executing the binary instruction sequence concurrently with the native architecture execution engine to produce a second final state;

emulating the binary instruction sequence according to the target architecture;

comparing the first final state and the second final state, wherein when the first final state and the second final state do not match, an emulation failure has occurred; and

enabling the emulation failure to be determined at a machine instruction level.